Creating Continuous Delivery for Yocto-based IoT Distribution

Alexander Kanevskiy

OpenIoT Summit Europe

2016-10-12

Who am I?

Alexander.Kanevskiy@Intel.com

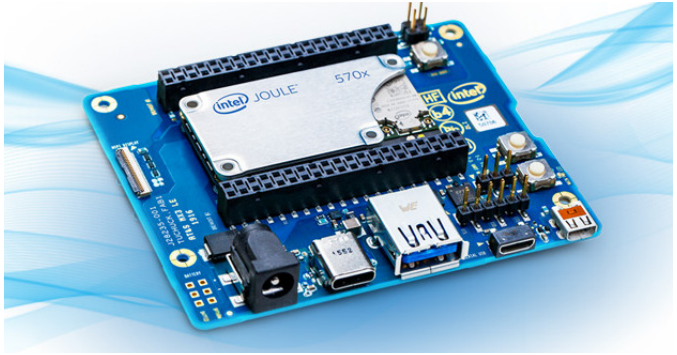
Agenda

- Yocto-based distributions: platforms and products
- Continuous Integration and Continuous Delivery 101
- Source Code Management
- Tools and practices for Platform and Product distributions
- Build and Automatic testing Infrastructure
- DevOps for CI/CD

Yocto-based distributions for Platform and Product

- Platforms
  - Combined from BitBake, OE-Core and additional layers
  - Poky-style, with generic features for multiple devices
    - https://github.com/ostroproject/ostro-os

- Products
  - Derivative distribution from Platform distribution
  - Small subset of device-specific settings and functionality. Example: Intel Joule™
    - https://github.com/ostroproject/ostro-os-xt



Continuous Integration and Continuous Delivery

Continuous Integration

- For every single change: build, automatically test, measure, visualize status
- Fix “broken” state as soon as possible.

2-Stage Continuous Integration

- Verify change(s) in sandbox before merging.
- Prevent “broken” state in your official branches

Continuous Delivery

- Make your releases “rolling”: ship software to your users in short iterations
- Make sure your “pipeline” has all steps that software needs to pass before shipment

Maximize throughput of **good** changes, **minimize breakages** of main code line

Source Code Management

Collaborative source code hosting solution is a **must** for effective CI/CD setup

- There are multiple options for public and private Git repositories
  - Cloud services and providers: GitHub, BitBucket
  - Self hosted/managed: GitLab, Gerrit, …
- Key “must have” features
  - Personal sandboxes for developers
  - Allow developers to easily collaborate
  - Change state tracking
  - Integration with external systems

Source Code Management: “Fork me on GitHub”

GitHub – de-facto standard in open source community nowadays

- Unlimited personal public repositories
- Reviews & Statuses
- APIs for integration with CI/CD
- Authentication and Access controls

Continuous Delivery for Platform Distributions

CI/CD for Platform Distributions: source code

Separate between your development and integration

- Layers
  - Development happening here
- Release Repository
  - Used only to integrate changes from layers
  - **The Release** of your software platform

Release Repository

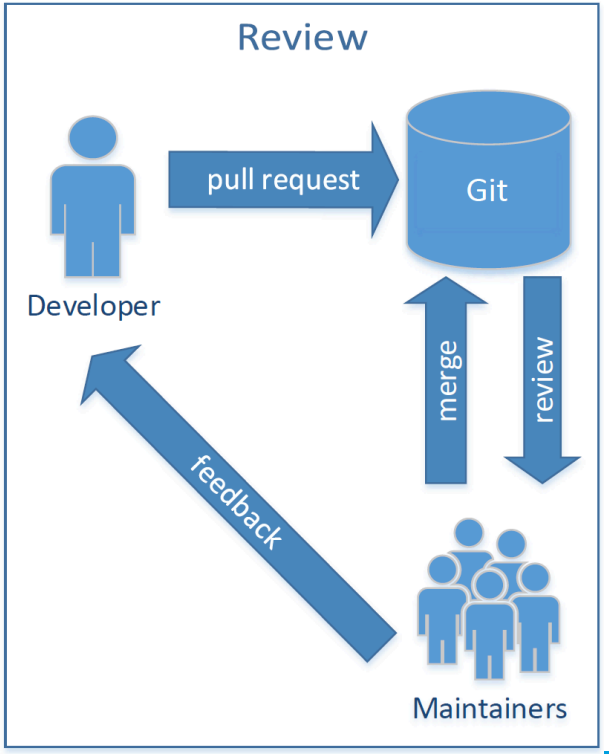
- The repository is constructed using **combo-layer** tool
  - combo-layer configuration files are the only content specific to release repository
  - CD is controlling promotion of changes from Layers to Release repository
  - Quality and schedule gates

```
[DEFAULT]
signoff = False

[bitbake]
src_uri = git://git.openembedded.org/bitbake
dest_dir = bitbake
hook = conf/combo-layerhook-generic.sh
branch = master
last_revision = 4bcf77589312d9936340d8c308006c2fc9baf67c

[openembedded-core]
src_uri = git://git.openembedded.org/openembedded-core
dest_dir = .
file_exclude = .templateconf
    README
hook = conf/combo-layerhook-openembedded-core.sh
branch = master
last_revision = cdaafc3729700778d95afc2413553d7b41c1317b

[meta-intel]
src_uri = git://git.yoctoproject.org/meta-intel
dest_dir = meta-intel
hook = conf/combo-layerhook-generic.sh
branch = master
last_revision = 7cceac01e1e0a5745600403d9b334babb76dc8ce

[meta-ostro]
src_uri = git@github.com:ostroproject/meta-ostro.git
dest_dir = .
file_exclude = .gitignore
hook = conf/combo-layerhook-generic.sh
branch = master
last_revision = eca9b627a3fee2d15812c17a79a14f8481094ad6

[meta-iotqa]
src_uri = git@github.com:ostroproject/meta-iotqa.git
dest_dir = meta-iotqa
hook = conf/combo-layerhook-generic.sh
branch = master
last_revision = 28618a57054583eeb290b78c48d2f8dd45675fb9
```

CI/CD for Layers

Provides service for layer maintainers and contributors

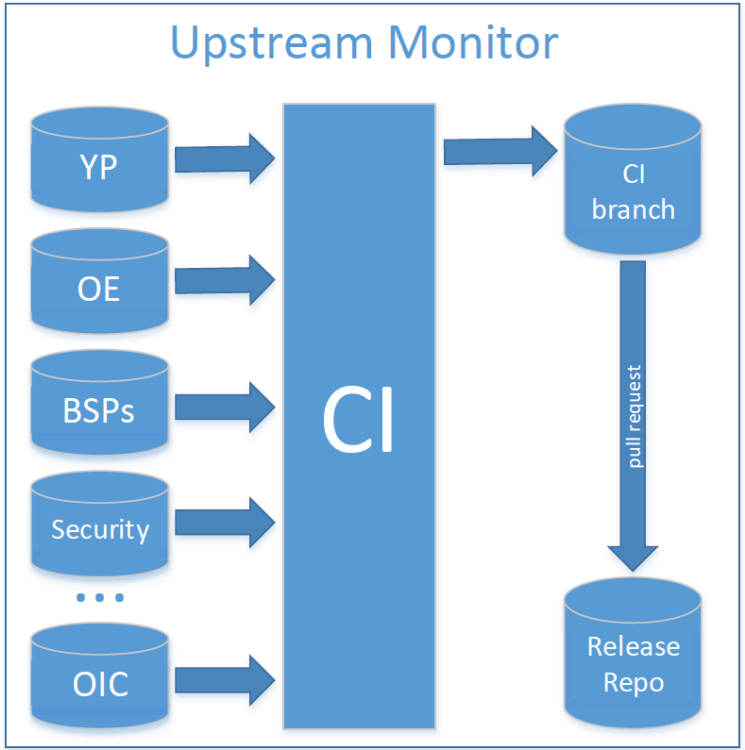
- Build every pull request on top of latest state of release repository and test on real hardware
- Build branch and merge head changes
- Provide feedback via GitHub commit status
- Watch for ”magic comments” by maintainers
  - This allows them to trigger builds using GitHub UI
- Security measures for PRs from “unknown” developers



CI/CD for Release repository

Release repository for official builds.

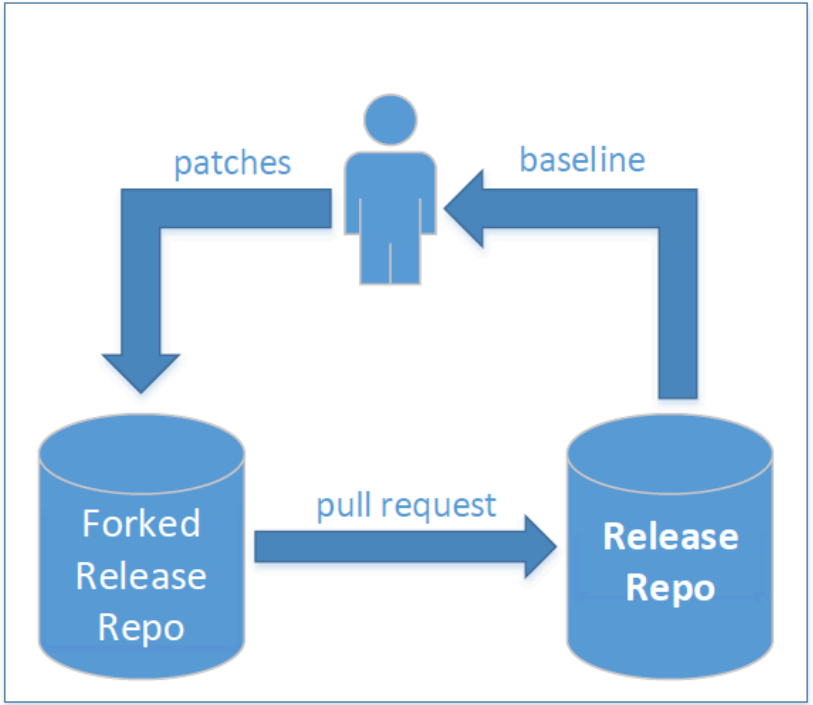
- Build on each merge to release branches
  - Builds are automatically tested on real hardware
  - Builds can be ”promoted” between stages of CD
- Pull Requests
  - Upstream Monitor creates or updates PR on every change of monitored branches in upstream layer repositories
  - Maintainers might override upstream monitor PRs in special cases
  - PRs from individual developers to release repository are ignored, except very special cases



Pull Requests to Release repository

When PRs to release repository are needed ?

- Introducing new layer to release repository
  - Update combo-layer.conf
  - Layer repository content would be imported on next upstream-monitor run
- Test builds for complex changes:
  - Changes across multiple layers that require orchestration
  - Tooling bug fixes are required
    - Bitbake
    - Classes from OE-core



CI/CD Engine: Jenkins

Open Source project with established and active community

- Newcomers friendly
- Extensible
- Scriptable
- DevOps friendly

CI/CD Implementation

For Platform Repository CI/CD is implemented in a “classical” way

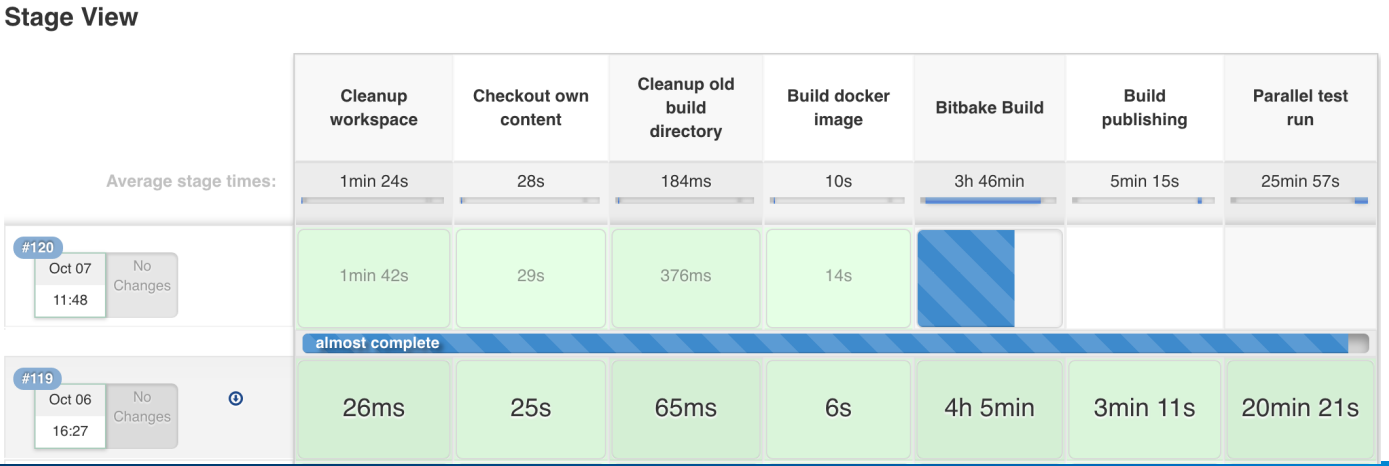
- Set of “freestyle” jobs to be triggered on events from GitHub
  - Orchestrator jobs, per layer or for release repository
    - layer_branch
    - layer_pull-request
  - Shared Build jobs and Test jobs
    - build_machine
    - test_hardware
- Post-processing jobs:
  - publishing, promotion, maintenance

Continuous Delivery for Product Distributions

Let’s use newer technologies

Jenkins 2.0: Pipelines as Code

- CD Pipeline can be shipped with code
- Easy scripting
- Support for multi-branch projects
- Easy parallel execution
- Persistent tasks during builds
- Integration with Docker



Pipeline as Code Inside Product Repository

Fine-grained control of process and resources inside your repository

- Jenkinsfile
- Scripts for building and testing
- Build and test targets

```groovy
#!groovy
def target_machine = "intel-corei7-64"
def test_devices = [ "joule", "minnowboardmax" ]
def build_os = "opensuse-42.1"
def current_project = "${env.JOB_NAME}".tokenize("_")[0]
def image_name = "${current_project}_build:${env.BUILD_TAG}"

node('docker') {
    ws ("workspace/builder-slot-${env.EXECUTOR_NUMBER}") {
        stage 'Cleanup workspace'
        deleteDir()

        stage 'Checkout own content'
        checkout poll: false, scm: scm
        stage 'Cleanup old build directory'
        dir('build') {
            deleteDir()
        }
        stage 'Build docker image'
        sh "docker build -t ${image_name} ${build_args} docker/${build_os}"
        def docker_image = docker.image(image_name)
        docker_image.inside(run_args) {
            stage 'Bitbake Build'
            sh "docker/build-project.sh"

            stage "Build publishing"
            sh "docker/publish-project.sh"
        }
        testinfo_data = readFile "${target_machine}.testinfo.csv"
    }
}

def test_runs = [:]
for(int i=0; i < test_devices.size(); i++) {
    def test_device = test_devices[i]
    test_runs["test_${test_device}"] = {
        node('ostro-tester') {
            writeFile file: 'tester-exec.sh', text: tester_script
            writeFile file: 'testinfo.csv', text: testinfo_data
            sh 'env && chmod a+x tester-exec.sh && ./tester-exec.sh'
    }
}
stage "Parallel test run"
parallel test_runs
```

CI/CD for Product Distribution

Few other different technologies

- Reproducible environment
  - Docker as build backend
  - Same default build targets for local and automated builds
- Using Git submodules instead of combo-layer approach
  - Distro configuration inheritance from base distribution

SCM: Git Submodules or combo-layer

combo-layer

- Pros
  - Self-contained repository
  - Easy to try any change in any place of repository
  - Easy to track individual upstream change
- Cons
  - Complexity of importing upstream repositories with non-linear Git history
  - Git history is polluted with upstream commit messages
  - Reviews might be time consuming, if there are many changes

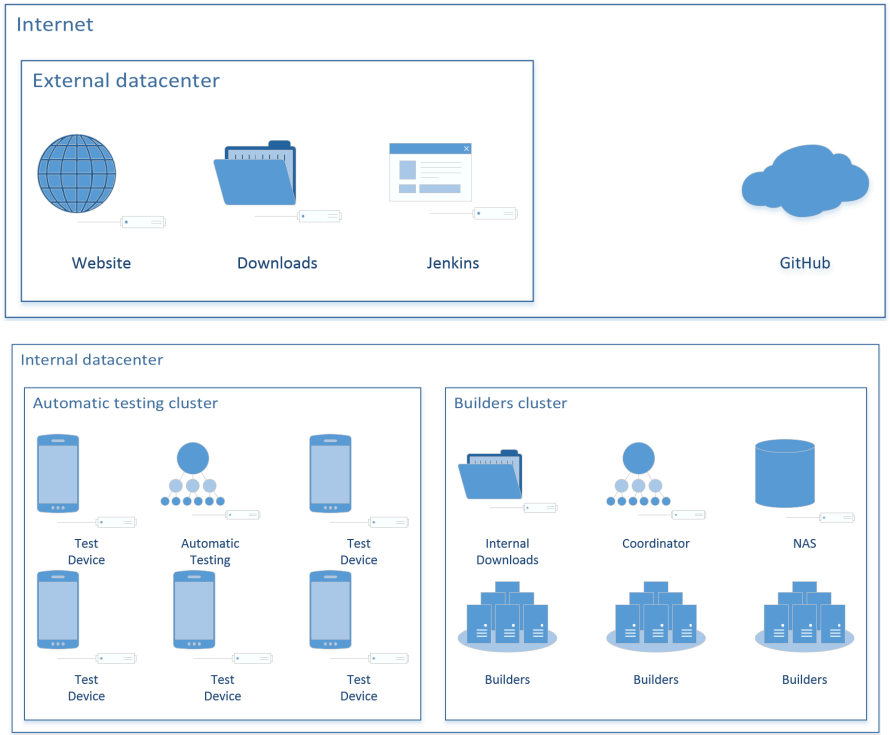
Git submodules

- Pros
  - Content from upstream repositories protected from accidental changes
  - Easier to maintain
  - Clean Git history, only product related changes are visible
- Cons
  - Dependency on upstream source hosting
  - Inability to quickly try changes for upstream components inside product builds
  - Harder to review upstream changes

Continuous Delivery Infrastructure

CI/CD Infrastructure: Architecture

- Frontend part
  - Web Site
  - Download server
  - Jenkins Master
- Isolated network for builders and testing
  - Coordinator
  - Network Storage
  - Builders
  - Automatic testing workers



DevOps for CI/CD

Maintain code of CI/CD pipeline same way as you develop your software

- Use Ansible to deploy host OS and configuration for services
  - Jenkins slave provisioning, Downloads, Network Storage, Docker installation
  - https://github.com/ostroproject/ostroproject-ci-ansible
- Jenkins Job DSL provisioning
  - https://github.com/ostroproject/ostroproject-ci
- Configure Jenkins to deploy build scripts to slave automatically
  - Production and Staging branches

DevOps: Jenkins initial seed

First and practically the only job configured manually in Jenkins.

And even this potentially can be done via Ansible.

- Simple pointer to git repository and branch with Groovy Job DSL script
  - Verifies that all needed plugins are present
  - Creates all other jobs
- To update your whole set of job in Jenkins, just push update into your CI repository

```
freeStyleJob("ci_seed_initial") {
    scm {
        git {
            remote {
                github(github_org+"/ostroproject-ci", protocol="https")
                credentials(credentials_github_https)
            }
            branches(ostro_ci_server)
        }
    }
    triggers {
        githubPush()
        scm(scm_poll)
    }
    steps {
        dsl {
            external('job-config/initial_seed.groovy')
            removeAction('DISABLE')
        }
    }
}
```

Generated Items:

- ci_deploy_scripts
- ci_seed_job_toplevel
- ci_seed_job_build
- ci_seed_job_test
- ci_cleanup_coordinator
- ci_cleanup_master
- ci_cleanup_worker
- ci_maintain_download_swupd_links
- ci_maintain_coordinator_swupd_links
- ci_seed_job_upstream_monitor
- ci_deploy_download_theme
- ci_deploy_documentation
- code_isafw_reports
- ci_seed_mirror_layers
- ci_seed_job_ostro-os-xt

DevOps: Jenkins Job DSL

Dynamically manage set of tasks for CI/CD

- Information from combo-layer configuration is used to pre-populate jobs
  - All layers and release repositories
  - Upstream Monitor
- Different seed jobs can handle multiple maintenance branches with different subsets of layers

Seed job: ci_seed_initial

Generated Items:

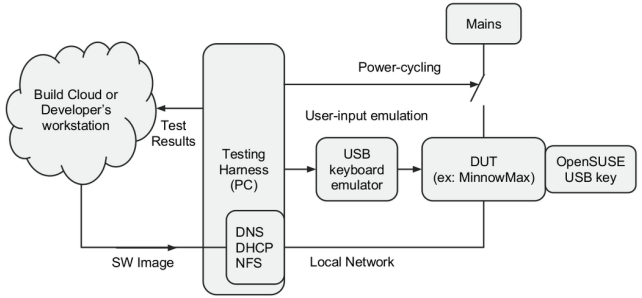
- meta-ostro_pull-requests
- meta-ostro_master
- meta-ostro-fixes_pull-requests
- meta-ostro-fixes_master
- meta-ostro-bsp_pull-requests
- meta-ostro-bsp_master
- meta-intel-iot-security_pull-requests
- meta-intel-iot-security_master
- meta-appfw_pull-requests
- meta-appfw_master
- meta-intel-iot-middleware_pull-requests
- meta-intel-iot-middleware_master
- meta-iotqa_pull-requests
- meta-iotqa_master
- meta-iot-web_pull-requests
- meta-iot-web_master
- meta-security-isafw_pull-requests
- meta-security-isafw_master
- meta-soletta_pull-requests
- meta-soletta_master
- ostro-os_pull-requests
- ostro-os_master
- tester-re-test-existing-build

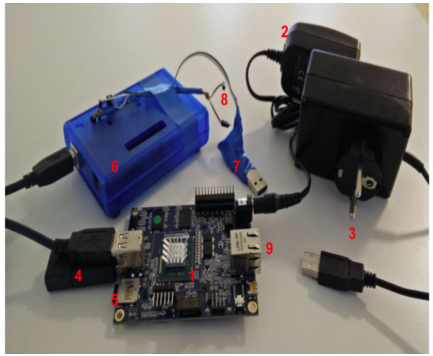
Other tools

Automatic Testing: https://github.com/01org/AFT

Reliable, Scalable and easy to Replicate framework to flash and execute test cases on physical HW

- Supports
  - PC-like devices, Edison, BeagleBone, MinnowBoard, Galileo, Joule
- Low cost:
  - Off-shelf components, <100$ BOM
- External test suite controls actual test execution
  - ELC 2015





2. Minnow Power Supply
3. USB-controlled power cutter.
4. OpenSUSE Thumb drive
5. SD Card (target media)
6. Arduino UNO R3
7. USB to Serial port Control interface for UNO
8. Programming toggle for UNO USB port.
9. Ethernet port

Miscellanea

- Keep buildhistory
  - Know what is in the build exactly: buildhistory-extra
  - Maintaining buildhistory for parallel builds
- Maintain S[shared]STATE
  - Local, over network, for PRs
- PRserver for parallel builds
- Performance and disk operations
- Benefits of using bmap-tools

Questions?

Links

- Reference:
  - Platform: https://github.com/ostroproject/ostro-os
  - Product: https://github.com/ostroproject/ostro-os-xt
  - CI/CD settings and scripts: https://github.com/ostroproject/ostroproject-ci
  - Ansible playbooks: https://github.com/ostroproject/ostroproject-ci-ansible
- Combo-Layer: https://wiki.yoctoproject.org/wiki/Combo-layer
- Jenkins Pipeline: https://jenkins.io/solutions/pipeline/
- Bmap-tools: https://github.com/01org/bmap-tools
- Automatic Flashing/Testing:
  - https://github.com/01org/AFT
  - https://github.com/ostroproject/meta-iotqa

Thank you!


intel®
experience
what's inside™

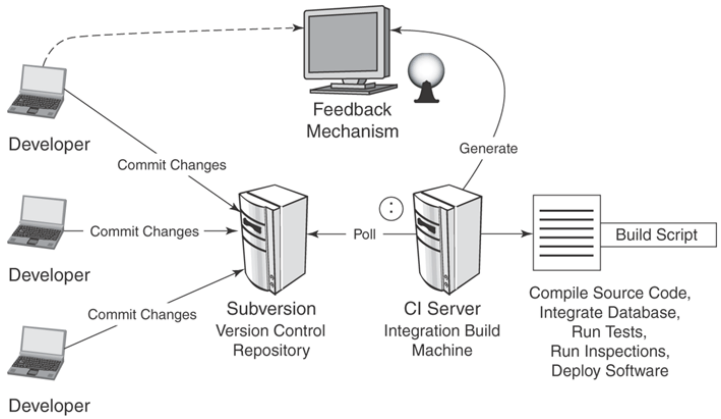
Appendix

Basic principles of CI and CD

Continuous Integration principles

Basic principles of CI

- Maintain code in version control repository
- Automate the build
- Make the build automatically tested
- Frequent integration to baseline
- Every commit to baseline should be built and tested
- Keep the build fast
- Make it easy to get the latest deliverables
- Anyone can see results of each build

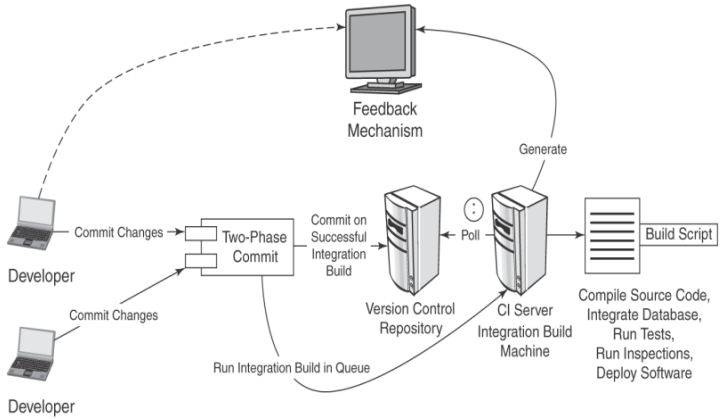


Source: Continuous Integration: Improving Software Quality and Reducing Risk. ISBN: 978-0-321-33638-5

Two-stage Continuous Integration principles

GitHub's pull request review mechanism can benefit with implementation of two-stage CI practices

- Every change committed to temporary place
- CI system perform build and test cycle
- When build and test results are good, change is merged to baseline

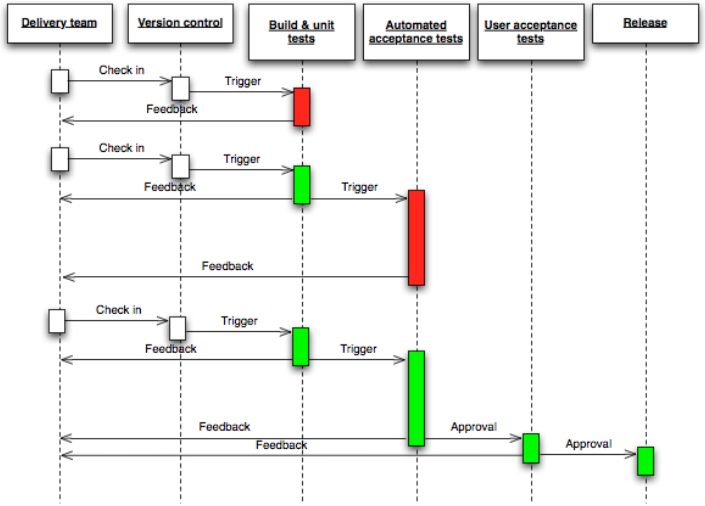


Source: Continuous Integration: Improving Software Quality and Reducing Risk. ISBN: 978-0-321-33638-5

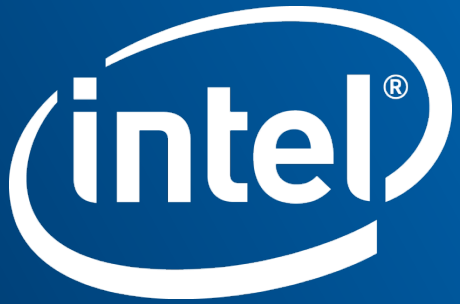
Continuous Delivery principles

Continuous Delivery practices allows us to improve product quality and produce predictable and reliable software releases often

- Set of validations through which a piece of software must pass on its way to release
- Tight integration with automated acceptance testing (BAT)
- Easy deployments to test environments
- Valuable software releases in short cycles
- Software reliably can be released at any time
- Fast way to produce bugfixes
- Any code commit may be released to customers at any point
- Feature toggles are useful for code which is not yet ready for use by end users



Source: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. ISBN: 978-0-321-60191-9


intel®